# Using Assume-Guarantee Contracts for Operational Verification of Autonomous Spacecraft

Pavan Rajagopal, CACI
James B. Dabney, University of Houston – Clear Lake
Julia M. Badger, NASA JSC

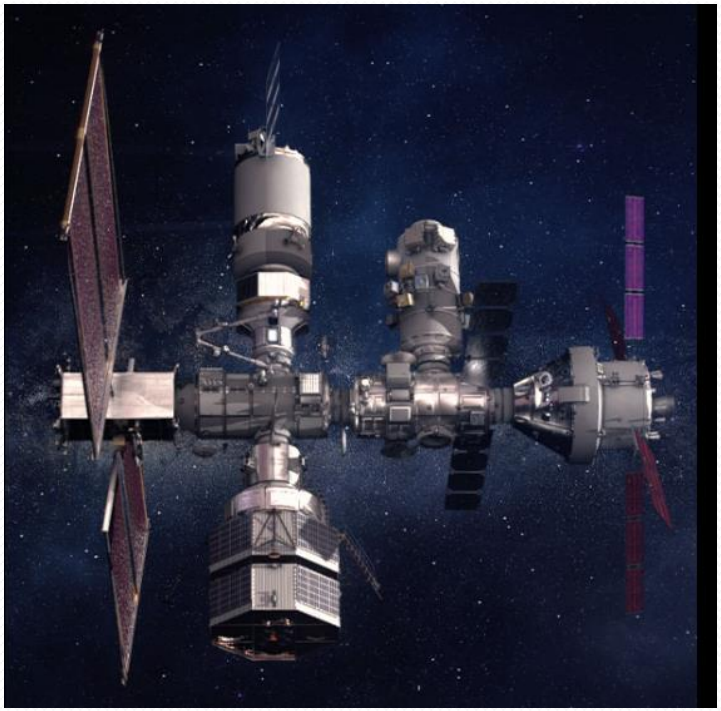
March 2023

# Overview

- Background & Motivation
- AGC definitions and format
- Defining/selecting operational contracts
- Implementing operational contracts
  - Inline
  - R2U2
- Workflow for testing and runtime environments
- Results and future work

# Lunar Gateway

- Artemis space station in cislunar (NRHO) orbit
- Gradual build up of modules
- Initial Co-Manifested Vehicle
  - Habitation and Logistics Outpost (HALO)
  - Power and Propulsion Element (PPE)
- Sustaining
  - International Habitation Module (IHAB)
  - Airlock
  - Visiting vehicles – Orion spacecraft

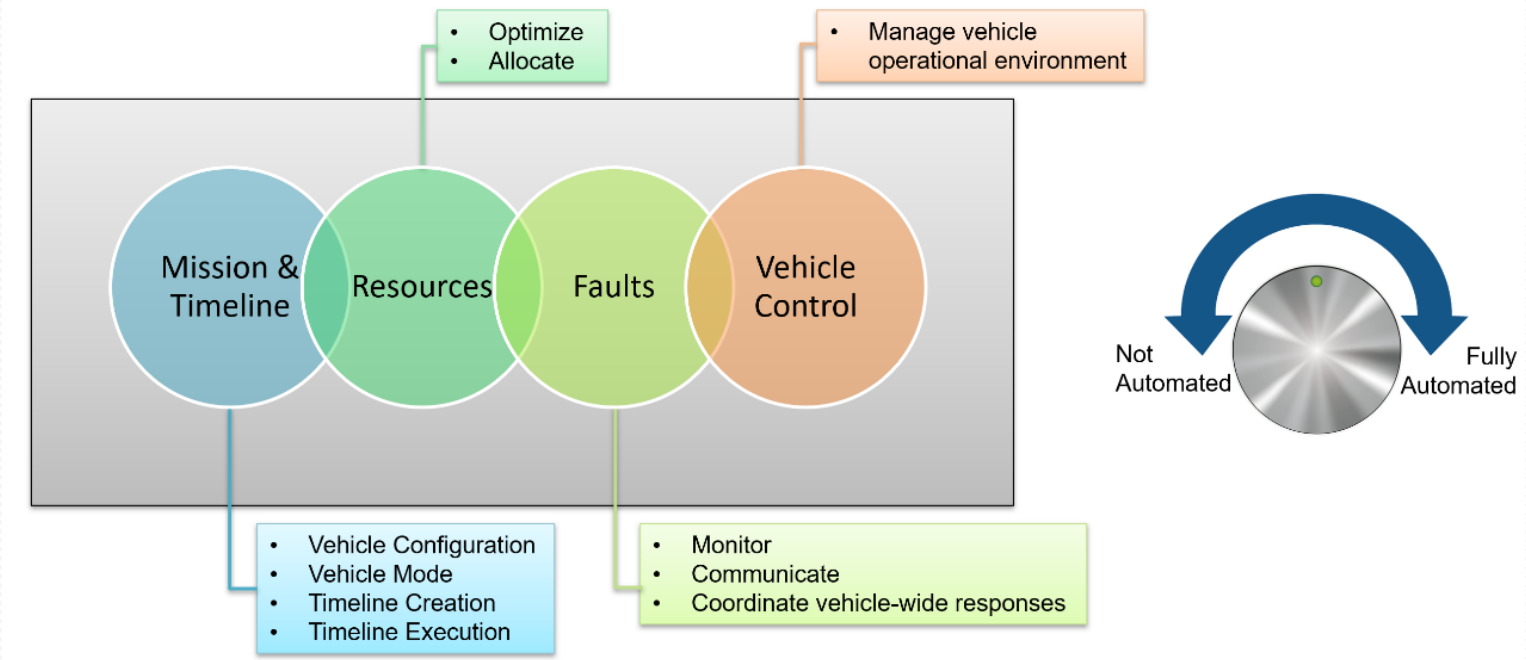


# Need for Autonomy

- Exploration spacecraft becoming more complex and autonomous
- Gateway Vehicle System Manager (VSM)
  - Manage unattended spacecraft for extended duration operations
  - Monitor and coordinate numerous subsystems through module system managers (MSMs)
  - Transition between varying levels of autonomy depending on crew and ground operator interaction

# Vehicle System Manager

- Four management functions
- Integrates modules; interfaces with humans and visiting vehicles
- Fully autonomous when uncrewed and no active ground control
- Can dial down autonomy to collaborate with flight crews and ground in operating vehicle



# Need for Enhanced Assurance

- VSM will encounter unexpected situations
  - Environmental anomalies
  - Unanticipated operational sequences
  - System failures
- VSM must stay within safe boundaries
  - Enforced during operation
  - Ensure safe recovery

# Assume-Guarantee Contracts

- AGCs define expected behavior of systems, subsystems, components
- Assumptions or pre-conditions on the subsystem's inputs bound the conditions in which the contract applies
- Guarantees or post-conditions of a subsystem's contract declare conditions that the subsystem's outputs must satisfy and can serve as verification goals
- Contracts can be used to define safety properties or hazard controls:
  - E.g a car should not start unless the transmission is in park mode
- Can be converted to a mathematical specification and model checked to ensure that specified system has the desired behavior and safety properties

# Defining AGCs

- Contract sources for VSM
  - Internal VSM contracts
    - Derived from software requirement specifications, scenarios / use cases
    - Watchdog-type functionality
  - MSM contracts – critical behavior of Gateway module system managers
    - Defined in MSM Interface Control Document (ICD)
    - MSM provider has primary responsibility for compliance
    - VSM must detect and react to anomalies at MSM interface

# Documenting AGCs (Template)

$C_1$:

- variables:
  - inputs: *values supplied to function*
  - outputs: *values returned from function*
- types: *data types (can be custom) of each input and output*
- assumptions: *conditions and constraints on inputs under which contract is valid*
- guarantees: conditions on outputs when contract is valid

# Selecting Contracts for Implementation

- Contracts are expressed mathematically
  - Propositional logic
  - Mission time Linear Temporal Logic
- Set of possible contracts is large and must be trimmed to feasible size
- Selection factors
  - Mission or safety-critical
  - Not covered by fault management
  - Contract has temporal element

# Development and Operational Verification

- Developmental verification
  - Performed at design time
  - Functionality represented as state machine using modeling language such as TLA+ or Promela
  - Contracts implemented as temporal formulas and compliance evaluated using model checker such as TLC or Spin
- Operational verification
  - Performed at runtime
  - Functionality is operational system
  - Contracts implemented as inline code or using runtime verification engine such as R2U2

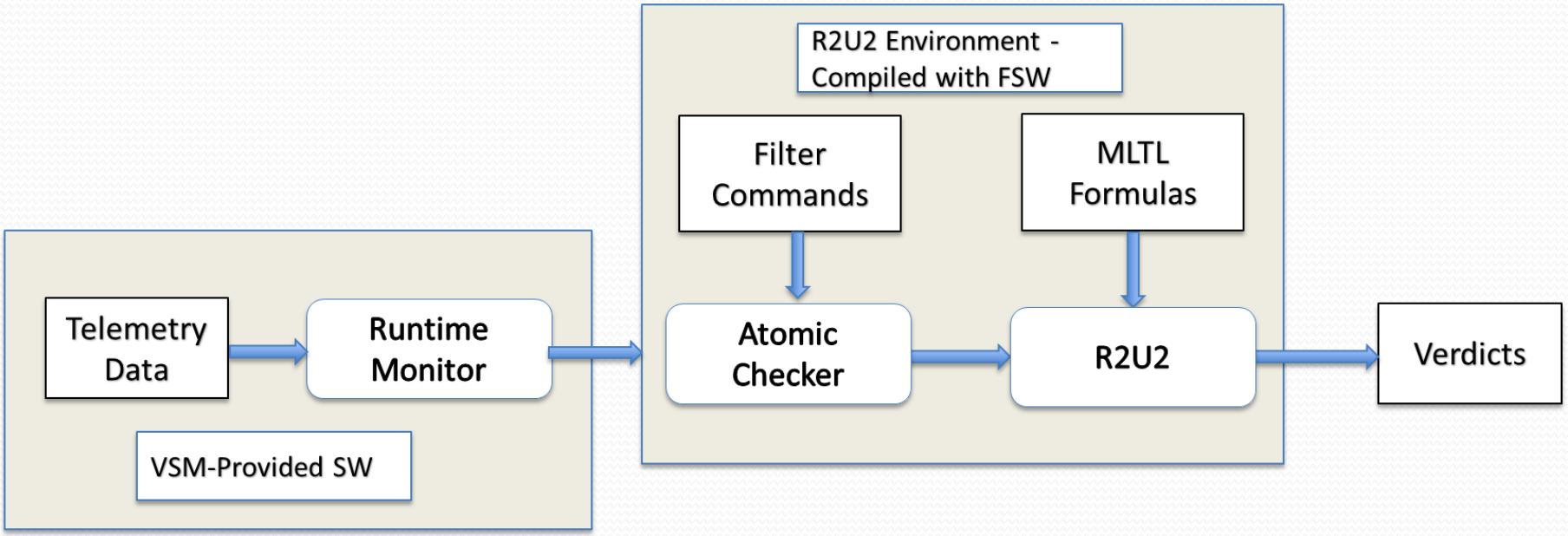
# Specifying Operational Contracts

- AGCs use two types of logic
  - Propositional: $A \wedge B$ *(A and B are both true now)*
  - Temporal: $\Diamond(A \wedge B)$ (*Eventually, A and B will both be true)*
- Both types can be coded in C, but
  - Coding temporal formulas is difficult
  - Coded temporal formulas are notoriously error-prone
  - Difficult to test – huge number of test cases required
  - Frequent source of verification and operational errors
- R2U2 provides a more reliable approach (next slide) using mission-time linear temporal logic
  - Adds time bounds to temporal relations

# R2U2 Advantages

- R2U2 formulas are written in math notation
  - Precise
  - Unambiguous
- Compact, optimized for size and speed
- R2U2 inferencing engine proven correct with respect to operator space https://research.temporallogic.org/papers/KZJZR20.pdf
- Correctness proof is the most complete form of verification, infeasible via testing https://www.researchgate.net/publication/3187522_The_Infeasibility_of_Quantifying_the_Reliability_of_Life-Critical_Real-Time_Software

# Operational Contract Implementation




- Runtime monitor (RuM) is interface between data sources and R2U2 system
- Atomic checker produces Booleans by filtering data (float, int) from RuM
- Atomic checker and R2U2 command file processed at compile time, generate executable code

# Workflow for Implementing & Testing

- Offline implementation & verification
  - Contracts specified in R2U2 MLTL file format
    - Atomic checker commands – convert input stream to booleans
    - MLTL specifications to document temporal formulas
  - Parser converts MLTL file into binaries that drive executable
  - Input stream simulated using csv format
  - R2U2 executable processes CSV, displays verdict stream
- Flight software (online) implementation
  - Runtime monitor (RuM) references same binaries
  - RuM extracts input stream from telemetry
  - Calls R2U2 library functions; passes input stream
  - Passes R2U2 verdict stream to Fault Manager for processing

# Results & Future Work

- Results
  - Example contracts implemented in multiple online environment (Linux, Target)
  - Implementation and verification workflows validated
  - Most contracts suitable for R2U2 require MLTL
- Future work
  - R2U2 team at Iowa State enhancing R2U2
    - More complex contracts including dynamic set operators
    - Runtime optimizations
    - Support tools
  - VSM team continuing to identify and implement contracts